Week 6 Lab

For an example of how to do the exercises you can refer to my week 6 GitHub repo which is at:

https://github.com/oit-gaden/Web-Development-2019-Winter/tree/master/Examples/Week6

Don't forget to push your code to GitHub in a folder called week6 and your Docker images (you will have 2) to Docker Hub.

Exercise 1 - Set up Postgres

1. Pull the Docker images for Postgres:

   docker pull postgres.

2. Download and installpgAdminfrom:

   https://www.pgadmin.org/

   For Mac users, you'll need to copy the pgAdmin4.app file to your applications folder.

Exercise 2 - Create the database with data

1. Create a Docker volume for storing the database data across container restarts by running the command:

   docker volume create db_data

2. Create a week 6 folder for this week's lab work in the usual place to be added to your GitHub repo.

3. Create a folder called database inside the week 6 folder.

4. Copy the docker-compose.yml file from my week6/database folder to the new database folder.

5. Start the Postgres container by opening a cmd/shell into your week6/database folder and running the command:

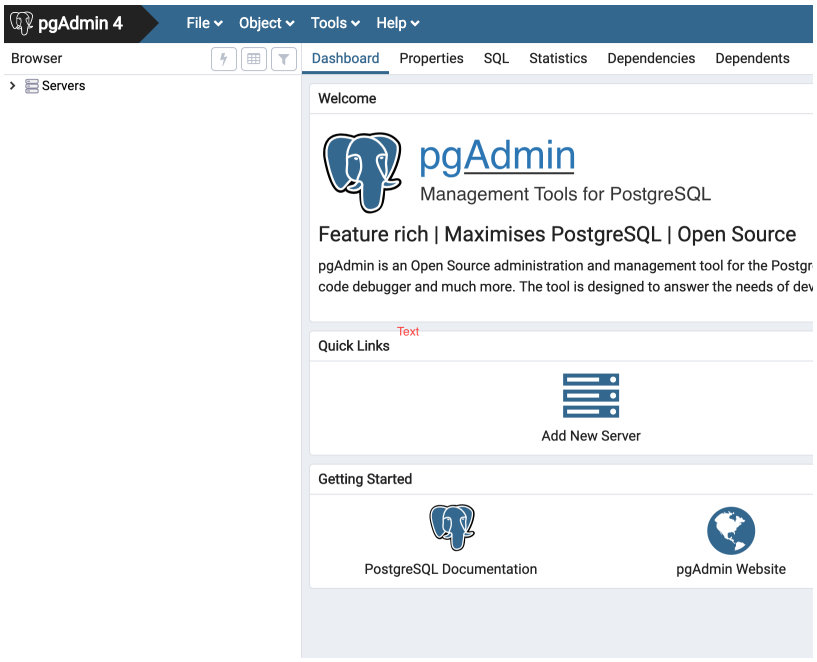
   docker-compose up -d

6. Run the command:

   docker ps

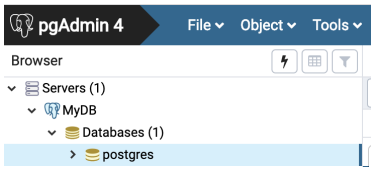
   to see what port the container is listening on. It should be 5432.

7. Start up **pgAdmin**. The UI will open as a tab in your default browser.



8. Right click on "Servers" in the left pane and select "Create -> Server". Give the server any name you want. For hostname/connection
in the "Connection" tab enter localhost for if you are using "Docker Desktop" and your Docker IP address if you are using Docker Toolbox.
Click on "Save". pgAdmin should now connect the Postgres database in your Docker container.

9. Navigate down the tree from "Servers" to where you see "Databases" as shown here:



10. Right click on **"Databases"** and select **"Create -> Database"**. Name the database **"School"**. Click on "Save".

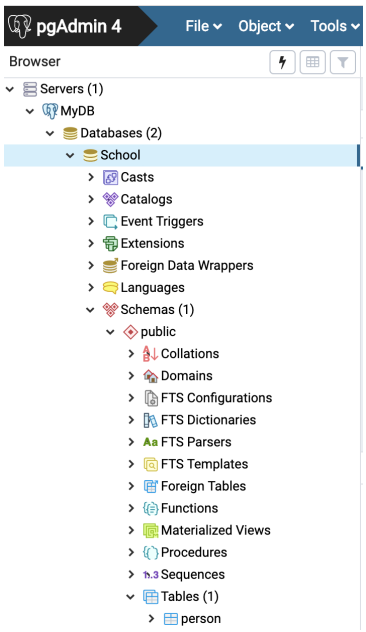
11. Click on the **"School"** database and pgAdmin should now "connect" to the "School" database.

12. Select "Query Tool" from the "Tools" menu.

13. Past in the script from my week6/database/person.sql file and click on the lightning bolt in the toolbar just above the query

window
to execute the create table command.

14. Navigate to and open "Tables" under the School database to see the newly created table as shown here:



15. Repeat steps 13-14 for the student.sql script. You'll need to right click on tables and select "refresh" to see the new student table.

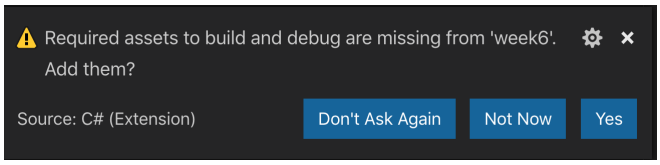
16. Execute person-data.sql script to populate the person table with data. Feel free to modify the data if you want before executing the script.

17. Execute the student-data.sql script to populate the student table with data. You'll need to query the person table to find the person primary
keys to use in the student-data.sql insert statements. Query the tables to make sure the data has been inserted.

Exercise 3 - Modify your REST service to now access the database

1. Copy the **webapi** folder from your week 5 folder to your new week 6 folder.

2. Open your week 6 folder in VSCode. Answer yes if you see the following dialog at the bottom right of VSCode:



3. In a cmd/shell window go to the webapi folder and run these commands:

   dotnet add package Npgsql

   dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL

   This will install Entity Framework for Postgres package.

4. Replace the your **Startup.cs** file with the one in my week6/webapi folder. Change the reference to **ECommerceDatabase** to**SchoolDatabase** in Startup.cs.

5. Create a Database folder like the one in my week6/webapi folder and mimic the code that I have for products and manufacturers in creating the data access for your students and persons. Hint: product would be similar to person as neither would have a foreign key reference while student would be similar to manufacturer as they both have foreign key references. Change the name of the DBContext class from ECommerceContext to SchoolContext anywhere it is referenced.

6. Create Controllers for both students and persons. The only method you need to implement is the one to get all of the entities from the database.

7. Add the ConnectionStrings setting to appSettings.Development.json file like this:

appsettings.Development.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  },
  "ConnectionStrings": {
    "ECommerceDatabase": "Host=localhost;Database=ecommerce;Username=postgres;Password="
  }
}
```

Change **ECommerceDatabase** to **SchoolDatabase** and**comme**
If you are using Docker Toolbox then use the discovered IP address rather than localhost.

8. Add the ConnectionSettings setting to appSettings.json file like this:

Change ECommerceDatabase to SchoolDatabase
and **ecommerce**to **School**.

9. Make sure your Postgres container is still running. Start the webapi by going to the Debug menu and selecting "Start without debugging".
Now use Postman to attempt to retrieve your students by using the URL: **http://localhost:5000/api/student** (IP address if you are using Docker Toolbox). Try retrieving your
person entries by using the
URL **http://localhost:5000/api/person**(IP address if you are using Docker Toolbox). Using **psgAdmin**, try inserting more data into your tables
and retrieving the data via Postman.

Exercise 4 - Use your webapp to retrieve the data from the database

1. Copy your webapp folder from week 5 to your week 6 folder.

2. Copy the docker-compose.yml file from my week6 folder.

3. In a cmd/shell window, go to your week 6 folder and run the command:

docker-compose up -d --build.

4. You should now be able to run your webapp in a browser using the URL http://localhost:8080 (like week 5) and access data now from your database.

5. Push your Docker images to Docker Hub and push your code to GitHub.

Extra Credit

Implement any or all of the additional CRUD functions in your application like my week 6 example. You will receive 20 lab points for each of the CRUD functions beyond get that you implement.